



This is part of **Family API** which allow to create dual-os version of program runs under OS/2 and DOS

Note: This is legacy API call. It is recommended to use 32-bit equivalent

2021/09/17 04:47 · prokushev · [0 Comments](#)

2021/08/20 03:18 · prokushev · [0 Comments](#)

DosSleep

This call suspends the current thread for a specified time. If the requested interval is 0, the call gives up the remainder of the current time slice.

Syntax

DosSleep (TimeInterval)

Parameters

- TimeInterval (ULONG) - input : Time interval in milliseconds until the thread is awakened.

Return Code

rc (USHORT) - return Return code descriptions are:

- 0 NO_ERROR
- 322 ERROR_TS_WAKEUP

Remarks

DosSleep suspends the current thread for the specified time period. The actual time it is asleep may be off by a clock tick or two, depending on the execution status of other threads running in the system.

If the time is 0, the thread gives up the remainder of the current time slice and allows any other ready threads of equal priority to run with the current thread for its next slice. Because the amount of sleep time specified is 0, an immediate return with 0 delay is made if no other ready thread is found. However, DosSleep does not yield to a thread of lower priority.

If the time is non-0, the time is rounded up to the resolution of the scheduler clock.

If DosSleep is used to regularly poll an external source to determine the occurrence of some event, a time equal to the longest response interval should be used.

For short time intervals, the rounding-up process combined with the thread priority interactions may cause a sleeping interval to be longer than requested. Also, when a process completes sleeping, it is scheduled for execution. But that execution could be delayed by hardware interrupts or by another thread running at a higher priority. A program should not use the `DosSleep` call as a substitute for a real-time clock because rounding of the sleep interval causes cumulative errors.

Asynchronous timers can be started with `DosTimerAsync` and `DosTimerStart`. `DosTimerAsync` starts a one-shot asynchronous timer, and `DosTimerStart` starts a periodic interval timer. `DosTimerStop` is issued to stop these timers.

Note: To ensure optimum performance, you should not use `DosSleep` in a single-thread Presentation Manager application. See `WinStartTimer`.

Family API Considerations

Some options operate differently in the DOS mode than in OS/2 mode. Therefore, the following restrictions apply to `DosSleep` when coding in DOS mode:

- `DosSleep` accuracy can be in error by 0.5%.
- `DosSleep` can degrade system performance of non-foreground program operations when DOS mode is in foreground.

Example Code

C Binding

```
#define INCL_DOSPROCESS

USHORT rc = DosSleep(TimeInterval);

ULONG TimeInterval; /* Interval size (in milliseconds) */

USHORT rc; /* return code */
```

The following example illustrates how to obtain the priority of a thread and how to change the priority. The main thread creates `Thread2` and allows it to begin executing. `Thread2` iterates through a loop that prints a line and then sleeps, relinquishing its time slice to the main thread. After one or two iterations by `Thread2`, the main thread obtains `Thread2`'s priority information and prints it. It then raises `Thread2`'s priority to fixed-high, and increments the level by ten. Since `Thread2` is now at a high priority, it immediately finishes its remaining iterations before relinquishing control on a long sleep; at this point, the main thread re-examines `Thread2`'s priority and reports its new priority level. In this example, it is helpful to understand how the `DosSleep` calls are used either to relinquish control of the processor, or to keep a thread alive (see `DosTimerAsync` or `DosTimerStart` for alternatives to `DosSleep`).

```
#define INCL_DOSPROCESS

#include <os2.h>
```

```

#define PRTYC_FIXEDHIGH 4 /* Priority class: fixed-high */
#define PRTY_DELTA 10 /* Priority delta: increase by 10 */
#define SEGSIZE 4000 /* Number of bytes requested in segment
*/
#define ALLOCFLAGS 0 /* Segment allocation flags - no sharing
*/
#define SLEEPSHORT 0L /* Sleep interval - 5 milliseconds */
#define SLEEPLONG 20L /* Sleep interval - 75 milliseconds */
#define RETURN_CODE 0 /* Return code for DosExit() */

VOID APIENTRY Thread2()
{
    USHORT i;

    /* Loop with four iterations */
    for(i=1; i<5; i++)
    {
        printf("In Thread2, i is now %d\n", i);

        /** Sleep to relinquish time slice to main thread **/
        DosSleep(SLEEPSHORT); /* Sleep interval */
    }
    DosExit(EXIT_THREAD, /* Action code - end a thread */
            RETURN_CODE); /* Return code */
}

main()
{
    USHORT Priority; /* Thread priority */
    USHORT Class; /* Priority class */
    USHORT Level; /* Priority level */
    SEL ThreadStackSel; /* Segment selector for thread stack */
    PBYTE StackEnd; /* Ptr. to end of thread stack */
    USHORT rc;

    /* Allocate segment for thread stack; this is better than just */
    /* declaring an array of bytes to use as a stack. Make pointer eos. */
    rc = DosAllocSeg(SEGSIZE, /* Number of bytes requested
*/
                    &ThreadStackSel, /* Segment selector
*/
                    ALLOCFLAGS); /* Allocation flags */
    StackEnd = MAKEP(ThreadStackSel, SEGSIZE-1);

    /* Start Thread2 */
    if(!(DosCreateThread((PFNTHREAD) Thread2, /* Thread address */
                        &ThreadID, /* Thread ID (returned) */
                        StackEnd))) /* End of thread stack */
        printf("Thread2 created.\n");
}

```

```

/** Sleep to allow Thread2 to execute */
if(!(DosSleep(SLEEPLONG)))                /* Sleep interval */
    printf("Slept a little to let Thread2 execute.\n");

/** Obtain Thread2's priority information and report it */
if(!(rc=DosGetPrty(PRTYS_THREAD,          /* Scope - single
                                           thread */
                   &Priority,             /* Address to put
                                           priority */
                   ThreadID)))            /* ID - thread ID */
{
    /* Extract priority class and level information */
    Class = HIBYTE(Priority);
    Level = LOBYTE(Priority);
    printf("Thread2: ID is %d, Priority Class is %d and Level is %d\n",
           ThreadID, Class, Level);
}
/** Raise Thread2's priority */
if(!(rc=DosSetPrty(PRTYS_THREAD,          /* Scope - single thread */
                   PRTYC_FIXEDHIGH,      /* Prty class - fixed-high */
                   PRTY_DELTA,           /* Prty delta - increase
                                           by 10 */
                   ThreadID)))            /* ID - thread ID */
{
    /* Obtain Thread2' new priority information and report it */
    rc=DosGetPrty(PRTYS_THREAD,          /* Scope - single thread */
                  &Priority,             /* Address to put
                                           priority */
                  ThreadID);             /* ID - thread ID */

    /* Extract priority class and level information */
    Class = HIBYTE(Priority);
    Level = LOBYTE(Priority);
    printf("Thread2: ID is %d, New Priority Class is %d and Level is %d\n",
           ThreadID, Class, Level);
}
}

```

MASM Binding

```

EXTRN  DosSleep:FAR
INCL_DOSPROCESS    EQU 1

PUSH   DWORD    TimeInterval ;Interval size (in milliseconds)
CALL   DosSleep

```

Returns WORD

Note

This text based on [http://www.edm2.com/index.php/DosSleep_\(FAPi\)](http://www.edm2.com/index.php/DosSleep_(FAPi))

Family API		
DOS	Process Manager	DosBeep DosExit DosSleep DosExecPgm
	File Manager	DosChDir DosChgFilePtr DosClose DosDelete DosDupHandle DosMkDir DosMove DosQCurDir DosQCurDisk DosSetFileMode DosOpen DosQFileInfo DosRead DosQFileMode DosQFSInfo DosQVerify DosRmdir DosSelectDisk DosFindClose DosFindFirst DosFindNext DosSetFileInfo DosSetVerify DosWrite DosFileLocks DosSetFHandState DosNewSize DosBufReset DosQFHandState DosSetFSinfo DosShutdown
	Memory Manager	DosFreeSeg DosSubAlloc DosSubFree DosSubSet DosAllocHuge DosAllocSeg DosReallocHuge DosReallocSeg DosGetHugeShift DosCreateCSAlias
	NLS	DosCaseMap DosGetCtryInfo DosGetDBCSEv DosSetCtryCode DosGetCollate DosGetMessage DosInsMessage DosPutMessage
	Date and Time	DosSetDateTime DosGetDateTime
	Devices	DosDevConfig DosDevIOct1 DosDevIOct2
	Signals	DosHoldSignal DosSetSigHandler
	Misc	BadDynLink DosGetEnv DosGetMachineMode DosGetVersion DosError DosErrClass DosSetVec
KBD	KbdCharIn KbdFlushBuffer KbdGetStatus KbdSetStatus KbdStringIn KbdPeek	
VIO	VioGetBuf VioGetConfig VioGetCurPos VioGetCurType VioGetPhysBuf VioReadCellStr VioReadCharStr VioScrollUp VioScrollDn VioScrollLf VioScrollRt VioScrUnLock VioSetCurPos VioSetCurType VioSetMode VioGetMode VioShowBuf VioWrtCellStr VioWrtCharStr VioWrtCharStrAtt VioWrtNAttr VioWrtNCell VioWrtNChar VioWrtTTY VioScrLock VioPopUp	
Tools	BIND	
Modules	DOSCALLS.DLL VIOCALLS.DLL KBDCALLS.DLL MSG.DLL	
Libraries	API.LIB OS2386.LIB FAPI.LIB DOSCALLS.LIB SUBCALLS.LIB	

2018/08/25 15:05 · prokushev · 0 Comments

From: <https://osfree.su/doku/> - **osFree wiki**

Permanent link: <https://osfree.su/doku/doku.php?id=en:docs:fapi:dossleep&rev=1629433539>

Last update: **2021/08/20 04:25**

